HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

MASTER THESIS

Mr./Mrs.
Mohammad Mohammadi

Hankel matrices for use in Learning Vector Quantization

2016

Faculty of **Applied Computer and Life Sciences**

MASTER THESIS

Hankel matrices for use in Learning Vector Quantization

Author:
Mohammad Mohammadi

Study Programme:
Applied Mathematics in Digital Media

Seminar Group:
MA14w1-M

First Referee:
Thomas Villmann

Second Referee:
Michael Biehl

Mittweida, September 2016

Bibliographic Information
Mohammadi, Mohammad: Hankel matrices for use in Learning Vector Quantization, 47 pages, 3 figures, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer and Life Sciences
Master Thesis, 2016

Abstract
Classification of time series has received an important amount of interest over the past years due to many real-life applications, such as environmental modeling, speech recognition, and computer vision.

In my thesis, I focus on classification of time series by LVQ classifiers. To learn a classifiers, we need a training set. In our case, every data point in the training set contains a sequence (an ordered set) of feature vectors. Thus, the first task is to construct a new feature vector (or matrix) for each sequence.

Inspired by [2], I use Hankel matrices to construct the new feature vectors. This choice comes from a basic assumption that each time series is generated by a single or a set of unknown Linear Time Invariant (LTI) systems.

After generating new feature vectors by Hankel matrices, I use two approaches to learn a classifier: Generalized Learning Vector Quntization (GLVQ) and Median variant of Generalized Learning Vector Quantization (mGLVQ).

I. Contents

II. List of Figures

III. List of Tables

IV. Preface

Classification is one of the most important tasks in machine learning. The goal is to learn a classifier from a training set. There are different types of classifiers. However, my thesis focuses on learning vector quantization (LVQ) classifiers, introduced by T. KOHONEN, as one the most intuitive prototype based classification models. In [21], you can find the developments of LVQ-variants for classification tasks in relation to several aspects of classification learning.

In a training set, a data point includes a feature vector and a label. The feature vector of the data point is supposed to be fixed. However, there are some situations that the feature vector of the data point changes during the time. It means there exists an ordered set (or time series) of feature vectors for each data point. Thus, we cannot use the methods described in [21].

To be able to use LVQ methods, we need to convert a set of feature vectors to one feature vector (or matrix). Inspired by Li [2], I use Hankel matrices to construct feature matrices, and then I will use LVQ to learn a classifier.

In chapter 1, I explained why Hankel matrices are used to construct feature vectors and then how to remove noise from the feature vectors. At last, I described two optimization approaches used in LVQ to find the best values of parameters.

In chapter 2, I presented two LVQ methods for classification. At first, I explained Generalized learning vector quantization (GLVQ) introduced by Sato and Yamada[22]. Then, I described the median variant of GLVQ which is introduced by D. Nebel[12].

In chapter 3, I combined the topics in chapter 1 and 2. From chapter 1, I can construct feature vectors, and then I can use the classification approaches, explained in chapter 2, to learn a classifier. In this chapter, I proposed two procedures for classification task.

In chapter 4, I illustrated three clustering approaches: Neural Gas, median variant Neural Gas and a median variant of k-means.

In chapter 5, I described a classification algorithm. I used Bag-of-Words (BoW) models to generate feature vectors called Bag-of-Hanklets (BoHk), then I used GLVQ to learn a classifier.

In chapter 6, I presented some results from implementation. I applied the algorithms, presented in chapter 3, for two data sets: Pen Digits and Libras.

At last, I would like to express my gratitude to my supervisor Professor Thomas Villmann, for his continued and valuable mentorship and Professor Michael Biehl, for the interesting topic. I am also grateful to the members of Computational Intelligence Group, specially David Nebel. A very special thanks goes out to my dear family and friends for their moral support.

1 Introduction

In machine learning, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs. Our prediction about a new observation is based on a training set of data containing observations (or instances) whose category membership is known.

In a typical classification problem, each observation in a training set has two parts

$$(x_i, l_i) \tag{1.1}$$

where

- x_i : a feature vector
- l_i : a label

However, I suppose the classification task for time series. It means each data point, instead of a single feature vector, includes an ordered set of feature vectors

$$(X_i, l_i) \tag{1.2}$$

where

- $X_i = (x_0, x_1, ..., x_T)$: an ordered set of feature vectors (time series)
- l_i : a label

Actually, this set of feature vectors represents a feature vector in which it changes during the time.

Hence, in order to classify a time series correctly, we have to discover the essence of the changes in the feature vector. So, we need a way to capture the temporal information of a time series.

Two important factors that have effect on the accuracy of prediction are:

1. The feature vectors that we use to classify
2. The dissimilarity measure to know the distance between feature vectors

According to them, there are two ways to capture temporal information:

- Some works use metrics for learning of temporal information: A new metric is presented in [10]. [9] uses Sobolev metrics.
- Another way is to use appropriate feature vectors. In my thesis I will use Hankel matrix to construct new feature vectors.

In the following sections, I will explain some introductions. In the first section, I will present the definition of Linear Time Invariant (LTI) systems, and then I will use it to explain why Hankel matrices are useful. In the second section, I will describe Principal Component Analysis (PCA) which is applicable to remove noise. In the last section, I will describe two optimization strategies which are so popular.

1.1 Dynamical Systems

Dynamical systems are powerful tools to work with temporally ordered sets because they can capture the essence of the temporal evolution of the time series.

In a dynamical system, there are two kinds of vectors:

- **State vector:** It is the input vector to the system, and it represent the internal state of the system. In our case, we suppose that we do not know anything about it.
- **Measurement vector:** It is the output vector of the system. In our case, it is the only thing we know about system, and according to the applications it represents different things. For example, it can be the coordinate of a tracked target at time k, or the pixel values of an images at time k.

As mentioned in [2], the goal of dynamical systems is to model temporal information of a sequence of a measurement vector $y_k \in \mathbb{R}^n$, as a function of a relatively low dimensional state vector $x_k \in \mathbb{R}^d$ that changes over time.

1.1.1 Linear Time Invariant (LTI) Systems

The simplest dynamical model is a linear time invariant (LTI) system.

Definition 1.1 A linear time invariant system is defined by two linear equations:

$$\begin{aligned} x_k &= Ax_{k-1} \qquad x_0 \ given \\ y_k &= Cx_k + w_k \end{aligned}$$

where matrices A and C are constant over time, and where w_k is uncorrelated zero mean Gaussian measurement noise.

The first equation is known as the state equation and involves the variable $x_k \in \mathbb{R}^d$, which represents the d-dimensional internal state of the LTI system. The second equation is known as the measurement equation and provides a link between the state of the system x_k and the n-dimensional observable measurement y_k.

Remark 1.2 In my thesis, I assume that

- Each time series is an output of a LTI system, and
- Time series in the same classes come from the same dynamical systems.

Hence, to classify a time series, we need to find out:

- Which dynamical system could generate the time series?

To answer this question, we need a method to specify systems. One way is to estimate the dimension and values of the matrices A and C and the initial vector x_0. However, given a finite number of measurements of y_k, the set of (A, C, x_0) that could have generated this data is not unique, and trying to jointly identify the dynamics (A, C) and x_0 leads to computationally challenging non-convex optimization problems [30].

Fortunately, Hankel matrices are a simple and powerful tool to tackle this problem.

1.1.2 Hankel Matrix

Definition 1.3 Given a sequence of output measurement vectors from the system, $y_0, y_1, ..., y_T$, its associated (block) Hankel matrix H_y is:

$$H_y = \begin{bmatrix} y_0 & y_1 & y_2 & \cdots & y_m \\ y_1 & y_2 & y_3 & \cdots & y_{m+1} \\ y_2 & y_3 & y_4 & \cdots & y_{m+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_k & y_{k+1} & y_{k+2} & \cdots & y_T \end{bmatrix}$$

where k is the maximal order of the system. T is the temporal length of the sequence, and it holds that $T = k + m - 1$.

Note that the columns of the Hankel matrix correspond to overlapping subsequences of the data, shifted by one time point, and that the block anti-diagonals of the matrix are constant. The following theorem [2] represents that this special structure of the matrix encapsulates the dynamic information of the system.

Theorem 1.4 *Let $\{y_k\}_k$ and $\{z_k\}_k$ be two trajectories of a system. The columns of two Hankel matrices H_y and H_z, corresponding to $\{y_k\}_k$ and $\{z_k\}_k$, span the same linear subspace, in the absence of noise.*

Proof: Let $w_k \equiv 0$. According to the definition of LTI systems, we can write:

$$y_k = Cx_k = CAx_{k-1} = CA^2x_{k-2} = ... = CA^kx_0 \tag{1.3}$$

So, the Hankel matrix can be represented as:

$$H_y = \Gamma X \tag{1.4}$$

where

$$\Gamma = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^k \end{bmatrix}$$

and

$$X = \begin{bmatrix} x_0 & x_1 & \cdots & x_m \end{bmatrix}$$

□

From the above formula, we get if the ranks of A and C are d,

$$f : \mathbb{R}^d \to \mathbb{R}^{kn \times d}$$
$$x_i \to \Gamma x_i$$

f is a linear transformation which maps each state vector x_i from $\mathbb{R}^d$ to a subspace of

$\mathbb{R}^{kn\times d}$ such that the dimension of this subspace is also d.

Corollary 1.5 *Regardless of the initial value, the columns of H_y and Γ span the same d-dimension subspace [2].*

If we would like to know whether two time series come from the same LTI system (and the same class) or not, we can easily compare their Hankel matrices (as feature matrices). If they are similar, we can say that they come from the same LTI system and they belong to the same class. Otherwise, it is concluded that they come from different LTI systems and they belong to different classes.

1.2 Principal Component Analysis (PCA)

The assumption underlying PCA is that the observed data are generated by a system that is driven by a relatively small number of latent (not directly observed) variables. The goal is to learn this latent structure [17].

Given a set of observation data, $x_i \in \mathbb{R}^n$, $i = 1, 2, ..., N$, of a random vector, PCA determines a subspace of dimension $d(\leq n)$, such that after projection on this subspace, the statistical variation of the data is optimally retained. This subspace is defined in term of d orthogonal axes, known as principle direction or principle axes, which are computed so that the variance of data, after projection on the subspace, is maximized.

If the mean of random variable X is zero, the principle axes are calculated in a step-wise way. First, suppose that $d = 1$ and the goal is to find a direction in $\mathbb{R}^n$ such that if the data is projected on it, the variance will be maximized.

Let u_1 denote the principle axis. The variance of the projections is given by

$$\sigma(u_1) = E\left[(u_1'X)(X'u_1)\right] = u_1'E(XX')u_1 = u_1'\Sigma u_1 \tag{1.5}$$

where E is the expectation operator and where Σ is the covariance matrix.

Now, the task is to maximize the variance. As we are only looking for a direction, let's suppose that u_1 is a unit vector. Thus the optimization task will be

$$u_1 = argmax_u \quad \left(u'\Sigma u\right) \qquad s.t. \quad u'u = 1 \tag{1.6}$$

Since this is a constrained optimization problem, we should use the method of Lagrange multipliers

$$L(u, \lambda) = u'\Sigma u - \lambda(u'u - 1) \tag{1.7}$$

Taking the gradient of L and setting it equal to zero we get

$$\Sigma u = \lambda u \tag{1.8}$$

So, the principle direction is an eigenvector of the covariance matrix, and we obtain

$$u_1 = argmax_u \quad \left(u'\Sigma u\right) = argmax_u \quad (\lambda)\,. \tag{1.9}$$

Hence, the variance is maximized if u_1 is the eigenvector that corresponds to the maximum eigenvalue, λ_1. Recall that, since the covariance matrix is symmetric and positive semidefinite, all the eigenvalues are real and nonnegative.

The second principle component is selected so that:

- is orthogonal to u_1 and
- maximizes the variance after projecting the data onto this direction

We should do the similar optimization task with an extra constraint, $u'u_1 = 0$. The second principle axis is the eigenvector corresponding to the second largest eigenvalue, λ_2.

The process continues until we obtain d principle axes; they are the eigenvectors corresponding to the d largest eigenvalues.

1.3 Optimization Approaches for functionals

In Learning Vector Quantization [21], which is focused in my thesis, the main question is

- What are the best values for parameters?

The cost functions help us to find the answer. The best places for prototypes are the ones that optimize the cost function. So, we need some method to optimize cost functions. Here, I will describe two methods which are so popular

- Stochastic Gradient Descent
- Expectation Maximization

According to the cost functions selected in problems, we choose one of them.

1.3.1 Stochastic Gradient Descent

Let

$$C(X, w) = \sum_{i=1}^{N} C_i(w) \tag{1.10}$$

be a cost function, where X, C_i and w are observations, i-th cost associated with i-th observation and parameters of the model, respectively.

Assume we would like to find a local minimum of the cost function by changing the parameters w. One question arises now

- In which direction will the cost function decrease the most?

Recall From Calculus

If a multi-variable function $F(x)$ is differentiable in a neighborhood of a point **a**, then $F(x)$ decreases fastest if one goes from **a** in the direction of the negative gradient of F at **a**, $\nabla F(a)$. It follows that, if

$$b = a - \gamma \nabla F(a) \tag{1.11}$$

for γ small enough, then $F(a) \geq F(b)$.

For the cost function, we derive

$$w^{new} = w^{old} - \gamma \nabla C(X, w^{old}) = w^{old} - \gamma \Sigma_{i=1}^{N} \nabla C_i(w^{old}) \tag{1.12}$$

This method is known as Gradient Descent learning.

In many cases, the summand functions have a simple form that enables inexpensive evaluations of the sum-function and the sum gradient.

However, in other cases, evaluating the sum-gradient may require expensive evaluations of the gradients from all summand functions. In this case, the true gradient of $C(X, w)$ is approximated by a gradient at a single example:

$$w = w - \gamma \bigtriangledown C_i(w) \tag{1.13}$$

As the algorithm sweeps through the training set randomly, it performs the above update for each training example. This is known as Stochastic Gradient Descent learning [29].

When the learning rates γ decrease with an appropriate rate, and subject to relatively mild assumptions, stochastic gradient descent converges almost surely to a local minimum [24].

1.3.2 Expectation Maximization (EM)

The expectation maximization algorithm is a general technique for finding maximum likelihood solutions for probabilistic models having latent variables.

In this section, I will describe the EM algorithm in two parts. At first, I will describe how the EM algorithm works, and I will give general idea and the algorithm. Then, I will present some proof. At last, I will talk about the difference the EM algorithm and the generalized EM algorithm applied in median variants of LVQ.

How the EM algorithm works

Let a log likelihood function be selected as a cost function

$$C(X, \Theta) = ln[p(X|\Theta)] \tag{1.14}$$

where X and Θ are the set of observations and parameters, respectively, and where $p(X|\Theta)$ is the likelihood function.

Now, the goal is to find the parameters so that they optimize the cost function. If p is a simple model, like Gaussian distribution, it will be easy to optimize the cost function. However, the accuracy of simple models are low. We have to hire some complex distributions to make the model more accurate.

One way to make more complex models is to combine several simple distributions. Here, the latent variables come to help us. The introduction of latent variables Z allows

complicated distributions to be formed from simpler components

$$p(X|\Theta) = \sum_{Z} p(X, Z|\Theta) \tag{1.15}$$

where Z is the set of all latent variables. The latent variables represent which component of the mixture distribution is responsible for a data point.

Now, the cost function will be

$$C(X, \Theta) = ln\{\sum_{Z} p(X, Z|\Theta)\} \tag{1.16}$$

A key observation is that the summation over the latent variables appears inside the logarithm. The presence of the sum prevents the logarithm from acting directly on the joint distribution, resulting in complicated expressions for the maximum likelihood solution. Hence, we need a way to prevent this problem.

To be more clear, let's use the terminology used in [15]. We shall call $\{X, Z\}$ the complete data set, and we shall refer to the actual observed data X as incomplete. The log likelihood function of the complete-data set is denoted by $ln[p(X, Z|\Theta)]$.

Maximization of this complete-data log likelihood function is straightforward. However, we do not have the values of the latent variables, and the only thing we can estimate is the posterior distribution of them. Since we cannot use the above log likelihood $ln[p(X, Z|\Theta)]$, we consider instead its expected value under the posterior distribution of the latent variable. The EM algorithm tries to maximize this expected value. It is depicted in [15]:

The EM algorithm

Given a joint distribution $p(X, Z|\Theta)$ over observed variables X and latent variables X, governed by parameters Θ, the goal is to maximize the likelihood function $p(X|\Theta)$ with respect to Θ

1. Choose an initial setting for the parameters Θ^{old}.
2. **E step** Evaluate the posterior distribution of latent variable $p(Z|X, \Theta^{old})$.
3. **M step** Evaluate Θ^{new} given by

$$\Theta^{new} = argmax_{\Theta}\zeta(\Theta, \Theta^{old}) \tag{1.17}$$

where

$$\zeta(\Theta, \Theta^{old}) = \sum_{Z} p(Z|X, \Theta^{old}) ln[p(X, Z|\Theta)]. \tag{1.18}$$

4. Check for convergence of either the log likelihood or the parameter values. If the convergence criterion is not satisfied, then let

$$\Theta^{old} \leftarrow \Theta^{new} \tag{1.19}$$

and return to step 2.

Why the EM algorithm works

In this section, I will explain why the EM is based on maximization of the expected value of complete-data log likelihood under the posterior distribution of the latent variable, instead of the incomplete-data log likelihood [15].

If we introduce a distribution $\gamma(Z)$ defined over the latent variables, then the following decomposition holds for any choice of $\gamma(Z)$

$$ln[p(X|\Theta)] = L(\gamma,\Theta) + KL(\gamma||p) \tag{1.20}$$

where

$$L(\gamma,\Theta) = \sum_Z \gamma(Z) ln\left(\frac{p(X,Z|\Theta)}{\gamma(Z)}\right)$$

$$KL(\gamma||p) = \sum_Z \gamma(Z) ln\left(\frac{p(Z|X,\Theta)}{\gamma(Z)}\right)$$

To prove the above formula, we should make use of the product rule for probabilities

$$lnp(X,Z|\Theta) = ln[p(Z|X,\Theta)] + ln[p(X|\Theta)]. \tag{1.21}$$

Substituting this into the expression for $L(\gamma,\Theta)$, the proof is completed.

As we can see $KL(\gamma||p)$ is the Kullback-Leibler divergence between $\gamma(Z)$ and the posterior distribution $p(Z|X,\Theta)$. I should mention that the Kullback-Leibler divergence satisfies $KL(\gamma||p) \geq 0$, with equality iff $\gamma(Z) = p(Z|X,\Theta)$. Therefore, $L(\gamma,\Theta)$ is a lower bound on $lnp(X|\Theta)$.

The EM algorithm tries to increase the lower bound of $L(\gamma,\Theta)$. Since the lower bound depends on $\gamma(Z)$ and Θ, the EM algorithm has two iterative stages, and it maximizes one of them in each stage.

In Expectation (or E) step, the current values of the parameters Θ^{old} are used to maximize the lower bound $L(\gamma,\Theta^{old})$ with respect to $\gamma(Z)$. Since $lnp(X|\Theta)$ does not depend on $\gamma(Z)$, the largest value of $L(\gamma,\Theta^{old})$ will occur when $KL(\gamma||p)$ vanishes. In other word, when $\gamma(Z)$ is equal to the posterior distribution $p(Z|X,\Theta^{old})$, $KL(\gamma||p)$ is equal to zero. In this case, the lower bound will be equal to the log likelihood.

In Maximization (or M) step, the distribution $\gamma(Z)$ is fixed and the lower bound $L(\gamma,\Theta)$ is maximized with respect to Θ (unlike E step) to give some new value Θ^{new}. This will increase the lower bound and also log likelihood function. Let's take a closer look:

- After the E step, the lower bound takes the form

$$L(\gamma,\Theta) = \sum_Z p(Z|X,\Theta^{old}) lnp(X,Z|\Theta) -$$

$$\sum_{Z} p(Z|X,\Theta^{old}) lnp(Z|X,\Theta^{old})$$

$$= \zeta(\Theta,\Theta^{old}) + const$$

Thus in the M step, the quantity that is being maximized is the expectation of the complete-data log likelihood.

The Generalized Expectation Maximization

In median variants of LVQ we use the Generalized Expectation Maximization (gEM,[23]) algorithm to optimize the cost function. The difference between the EM and the gEM is on the maximization step.

In the maximization step of the gEM algorithm we do not look for new parameters which maximize the ζ function. We only need to find new parameters so that the ζ function is increased.

The gEM algorithm is depicted in the following:

The EM algorithm

Given a joint distribution $p(X,Z|\Theta)$ over observed variables X and latent variables X, governed by parameters Θ, the goal is to maximize the likelihood function $p(X|\Theta)$ with respect to Θ

1. $t = 0$.
2. Choose an initial setting for the parameters Θ^{old}.
3. **E step** Evaluate the posterior distribution of latent variable $p(Z|X,\Theta^{old})$.
4. **gM step** Evaluate Θ^{new} in such a way that

$$\zeta(\gamma,\Theta^{new}) > \zeta(\gamma,\Theta^{old}) \tag{1.22}$$

   where

$$\zeta(\Theta,\Theta^{old}) = \sum_{Z} p(Z|X,\Theta^{old}) lnp(X,Z|\Theta). \tag{1.23}$$

   if searching for new parameters failed, then set $\Theta^{new} = \Theta^{old}$.
5. $t = t+1$.
6. If $t < NumberOfIteration$, then let

$$\Theta^{old} \leftarrow \Theta^{new} \tag{1.24}$$

   and return to step 3. Otherwise, End.

2 The Learning Vector Quantization (LVQ) for Prototype Based Classification

Let $X \subset R^n$ and $L = \{1, 2, ..., k\}$ be data and label spaces, respectively. Suppose $\{(x_i, l_i) | x_i \in X, l_i \in L, i = 1, 2, ..., N\}$ be a training set, where x_i is data point and l_i is its associated label.

In LVQ, there are some prototypes $\theta_i \in \Theta, i = 1, ..., M$ with the labels $c_i \in L, i = 1, ..., M$, and we would like to distribute them in data space as well as possible, i.e. when we use these prototypes for prediction, fewer errors happen. After learning prototypes, we use them to predict the labels of new data points in such a way that the label of a new data point is assumed to be equal to the label of the Nearest Prototype.

Here, I will focus on two LVQ algorithms for learning the prototypes

- The Generalized Learning Vector Quantization (GLVQ) [22]
- The Median-Generalized Learning Vector Quantization (mGLVQ) [11],[12]

They have different cost functions

- Based on the number of misclassifications
- Based on a likelihood function

By optimizing the cost function, the appropriate positions for prototypes can be found.

For the first case, Stochastic Gradient Descent learning can be used to minimize the cost function. Hence, we should calculate the derivative of the cost function with respect to prototypes.

For the second case, the mGLVQ method uses the generalized Expectation Maximization (gEM) algorithm to maximize the likelihood function.

In the following sections, I will describe GLVQ and mGLVQ, respectively.

2.1 Generalized Learning Vector Quantization

Let's define the cost function as:

- the number of misclassifications in the training set

To find the cost value, we should count the number of misclassifications.

We define the distances $d^+(x_i)$ and $d^-(x_i)$ as

$$d^+(x_i) = min_{\{\theta_j | y_i = c_j\}} d(x_i, \theta_j) \tag{2.1}$$

$$d^-(x_i) = min_{\{\theta_j | y_i \neq c_j\}} d(x_i, \theta_j) \tag{2.2}$$

where d^+ represents the minimal distances from x_i to the closest prototype θ^+ of

the same class (correct) and where d^- represents the minimal distances from x_i to the closest prototype θ^- of the different classes (incorrect). By these values, we define the classifier function

$$\mu(x_i) = \frac{d^+(x_i) - d^-(x_i)}{d^-(x_i) + d^+(x_i)} \tag{2.3}$$

According to this definition, a data point x_i is misclassified iff $\mu(x_i) > 0$ is valid (or $d^+(x_i) > d^-(x_i)$).

Now, we use the classifier function to count the number of misclassifications (cost function):

$$C(X, \Theta) = \sum_{i=1}^{N} H(\mu(x_i)) \tag{2.4}$$

where $H(.)$ is the Heaviside function.

To optimize a function by the gradient descent algorithm, we need a differentiable function. But the Heaviside function is not differentiable if $\mu(x) = 0$ and everywhere else the gradient is zero.

The GLVQ algorithm approximates the cost function by a differentiable cost function such that gradient descent learning becomes available. The Heaviside function is replaced by sigmoid function, which is defined as

$$sgd(x) = \frac{1}{1 + exp(-x)}$$

Hence, the cost function will be

$$C(X, \Theta) = \sum_{i=1}^{N} sgd(\mu(x_i)) \tag{2.5}$$

Now, everything is ready: we have a differentiable cost function, and we can minimize this function by the stochastic gradient descent learning.

For a given data point x_i, the update for the prototypes $\theta^{\pm}$ becomes

$$\triangle \theta^{\pm} = \eta \frac{\partial_s sgd(\mu(x_i))}{\partial \theta^{\pm}} \tag{2.6}$$

where η is the learning rate. The full derivative is:

$$\triangle \theta^{\pm} = \eta . \frac{\partial sgd(\mu(x_i))}{\partial \mu(x_i)} . \frac{\pm 2 d^{\mp}(x_i)}{(d^+(x_i) + d^-(x_i))^2} . \frac{\partial d^{\pm}(x_i)}{\partial \theta^{\pm}} \tag{2.7}$$

Hence, in order to use the GLVQ, we need to apply a differentiable dissmilarity measure, and we have to find the derivatives of the dissimilarity measure with respect to the prototypes.

2.2 The Median Generalized Learning Vector Quantization (mGLVQ)

The median variants of LVQ require only dissimilarities between all pairs of data points to be known. Further, the prototypes are restricted to be data points, and we should select prototypes from data points such that they optimize the cost function. Since the number of data points are finite, it corresponds to a discrete optimization problem [11].

The advantage of the median approaches is that they are still applicable if:

- the data have complicated structures and / or
- the dissimilarity measure is not differentiable (so we cannot use stochastic gradient algorithms)

Here, I will focus on the median variant of generalized learning vector quantization [12]. In [11], you can find more median variants of LVQ.

2.2.1 Median GLVQ

Here, we follow [11] and [12]. As the cost function of GLVQ is introduced in equation (2.5), the classifier function μ only depends on the two winners of x_i (one with the same label and another with different label).

For mGLVQ, we would like to follow the same idea, i.e. the cost function only depends on the two winners (one with the same label and another with different label) of data points

$$C_{mGLVQ}(X,\Theta) = \sum_{i=1}^{N} log(g^+(x_i,\Theta) + g^-(x_i,\Theta)) \tag{2.8}$$

where g^+ and g^- are defined as

$$g^+(x_i,\Theta) = \alpha/2 - \frac{d^+(x_i)}{d^+(x_i)+d^-(x_i)} \tag{2.9}$$

$$g^-(x_i,\Theta) = \alpha/2 + \frac{d^-(x_i)}{d^+(x_i)+d^-(x_i)} \tag{2.10}$$

Hence, we still have a deterministic view. To make applicable the gEM algorithm for the cost function, formal probabilities are introduced as

$$p^+(\Theta|x_i) = \frac{g^+(x_i,\Theta)}{g^+(x_i,\Theta)+g^-(x_i,\Theta)} \tag{2.11}$$

$$p^-(\Theta|x_i) = \frac{g^-(x_i,\Theta)}{g^+(x_i,\Theta)+g^-(x_i,\Theta)} \tag{2.12}$$

which sum up to $p^+(\Theta|x_i) + p^-(\Theta|x_i) = 1$. Actually, these formal probabilities just play the role of the probabilities $p(\theta_i|x_i,l_i)$ in the EM/gEM algorithms.

In addition, we define the quantities $\gamma^+(\Theta|x_i) \geq 0$ and $\gamma^-(\Theta|x_i) \geq 0$ which plays the role of γ in the EM/gEM algorithm, so it sum up to

$$\gamma^+(\Theta|x_i) + \gamma^-(\Theta|x_i) = 1 \tag{2.13}$$

The respective Kullback-Leibler-divergence for each x_i is now defined as

$$KL_i(\gamma||\Theta) = \gamma^+(\Theta|x_i).log(\frac{\gamma^+(\Theta|x_i)}{p^+(\Theta|x_i)}) + \gamma^-(\Theta|x_i).log(\frac{\gamma^-(\Theta|x_i)}{p^-(\Theta|x_i)}) \tag{2.14}$$

and the loss term for each x_i is obtained as

$$L_i(\gamma, \Theta) = \gamma^+(\Theta|x_i).log(\frac{g^+(x_i, \Theta)}{\gamma^+(\Theta|x_i)}) + \gamma^-(\Theta|x_i).log(\frac{g^-(x_i, \Theta)}{\gamma^-(\Theta|x_i)}) \tag{2.15}$$

From the EM/gEM algorithm, equation (1.20), we get

$$C(X, \Theta) = L(\gamma, \Theta) + KL(\gamma||p) \tag{2.16}$$

where

$$L(\gamma, \Theta) = \sum_{i=1}^{N} L_i(\gamma, \Theta) \tag{2.17}$$

$$KL(\gamma||p) = \sum_{i=1}^{N} KL_i(\gamma||p) \tag{2.18}$$

We can prove this statement by the following calculations:

Introducing the abbreviations

$$\begin{array}{ll} g_i^+ = g^+(x_i, \Theta) & g_i^- = g^-(x_i, \Theta) \\ \gamma_i^+ = \gamma^+(\Theta|x_i) & \gamma_i^- = \gamma^-(\Theta|x_i) \end{array}$$

From the equations (2.8) and (2.13), we get

$$C(X, \Theta) = \sum_{i=1}^{N} log\left(g^+(x_i, \Theta) + g^-(x_i, \Theta)\right)$$

$$= -\sum_{i=1}^{N} \left(\gamma_i^+ + \gamma_i^-\right) log\left(\frac{1}{g_i^+ + g_i^-}\right)$$

$$= -\sum_{i=1}^{N} \left(\gamma_i^+ + \gamma_i^-\right) log\left(\frac{1}{g_i^+ + g_i^-}\right) + \sum_{i=1}^{N} \gamma_i^+ log\left(\frac{g_i^+}{\gamma_i^+}\right) +$$

$$\sum_{i=1}^{N} \gamma_i^- log\left(\frac{g_i^-}{\gamma_i^-}\right) - \sum_{i=1}^{N} \gamma_i^+ log\left(\frac{g_i^+}{\gamma_i^+}\right) - \sum_{i=1}^{N} \gamma_i^- log\left(\frac{g_i^-}{\gamma_i^-}\right)$$

$$= \sum_{i=1}^{N} \gamma_i^+ log\left(\frac{g_i^+}{\gamma_i^+}\right) + \sum_{i=1}^{N} \gamma_i^- log\left(\frac{g_i^-}{\gamma_i^-}\right) -$$

$$\sum_{i=1}^{N} \gamma_i^+ log\left(\frac{g_i^+}{\gamma_i^+(g_i^+ + g_i^-)}\right) - \sum_{i=1}^{N} \gamma_i^- log\left(\frac{g_i^-}{\gamma_i^-(g_i^+ + g_i^-)}\right)$$

$$= \sum_{i=1}^{N} \gamma_i^+ log\left(\frac{g_i^+}{\gamma_i^+}\right) + \sum_{i=1}^{N} \gamma_i^- log\left(\frac{g_i^-}{\gamma_i^-}\right) - \sum_{i=1}^{N} \gamma_i^+ log\left(\frac{p_i^+}{\gamma_i^+}\right) -$$

$$\sum_{i=1}^{N} \gamma_i^- log\left(\frac{p_i^-}{\gamma_i^-}\right) = \sum_{i=1}^{N} L_i(\gamma, \Theta) + \sum_{i=1}^{N} KL_i(\gamma||p)$$

Hence, we get

$$C_{mGLVQ}(X, \Theta) = \sum_{i=1}^{N} L_i(\gamma, \Theta) + \sum_{i=1}^{N} KL_i(\gamma||p)$$

Now, we can use gEM algorithm. In particular, the E-step consists in the calculations

$$\gamma^+(\Theta, x_i) = \frac{g^+(x_i, \Theta)}{g^+(x_i, \Theta) + g^-(x_i, \Theta)} \tag{2.19}$$

$$\gamma^-(\Theta, x_i) = \frac{g^-(x_i, \Theta)}{g^+(x_i, \Theta) + g^-(x_i, \Theta)} \tag{2.20}$$

In gM-step, we search a data point x_l so that the lower bound is improved

$$L(\gamma, \Theta^{new}) > L(\gamma, \Theta) \tag{2.21}$$

where Θ^{new} is obtained from Θ taking the new assignment $\theta_j \leftarrow x_l$.

3 Using Dynamic Subspace Angles

To exploit the temporal information encoded in the data, we use dynamical system approach, i.e. we assume that the time series are outputs of unknown dynamic systems. Particularly, we suppose that

- the time series is an output trajectory of an underlying, unknown LTI system.

In this context, different realizations of a class corresponds to trajectories of the same system in response to different initial conditions. The columns of their Hankel matrices span the same subspace, as explained before in chapter 1.

This allows us to measure the similarity between two time series by simply measuring the similarity between the associated subspace.

In the following sections, at first I will describe how to use PCA to remove noise in time series. In the second section, I will introduce some measures to compute dissimilarities between two subspaces. At last, two algorithms will be given to train a classifier for time series.

3.1 Principal Component Analysis for Noise Reduction

The algorithms, which later presented in this chapter, use PCA. The reasons come from two aspects:

1. Statistical aspect: In our case, the elements of Hankel matrices are linearly correlated, and PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables.
2. Noise aspect: As I suppose that each time series is an output of an unknown LTI system such that the dimension of the state space is d, the rank of its associated Hankel matrix will be d, in absence of noise. However, the noise will increase the rank of Hankel matrix. So, we need to reduce the noise.

Since the subspaces, generated by the columns of Hankel matrices of time series of a certain class, are equal, I will suppose that each column of Hankel matrices is a vector, and I will use PCA

$$HH' \approx P\Lambda P' \tag{3.1}$$

where P and Λ are the eigenvalue and eigenvector matrices of the d largest eigenvalues, respectively. Now, I get an orthogonal basis P for the subspace generated by Hankel matrix. I will use this orthogonal basis as a feature vector (matrices) for training.

3.2 Similarity Measures of Linear Subspaces

If we would like to compare two time series whether they come from the same class or not, we need to compare the subspaces generated by their Hankel matrices. So we need

some similarity or dissimilarity measures.

The canonical correlations [1] measure the angles between the closest vectors from two subspaces.

- A high canonical correlation value corresponds to a small subspace angle and to subspaces that are close to each other.
- A small canonical correlation corresponds to a subspace angle near $\pi/2$ or subspaces that are close to be orthogonal.

Thus, in classification applications, classes that have higher canonical correlations are more separated and easier to discriminate.

3.2.1 Canonical Correlation

Definition 3.1 Given two linear subspaces F and G such that

$$p = dim(F) \geq dim(G) = q \geq 1, \tag{3.2}$$

Canonical correlations are cosines of principal angles $0 \leq \theta_1 \leq \theta_2 \leq ... \leq \theta_q \leq \pi/2$, which are defined as:

$$cos\theta_k = max_{u_k \in F} \left[max_{v_k \in G} \left(u_k^T v_k \right) \right] \quad ||u_k||_2 = ||v_k||_2 = 1 \tag{3.3}$$

subject to $u_i^T u_i = v_i^T v_i = 1$, $u_i^T u_j = v_i^T v_j = 0, i \neq j$ [1].

In a simple words, principal angles are defined as:

- θ_1 is the smallest angle that exists between two vectors, one from F and another from G.
- Assuming that θ_1 is angle between u_1 and u_2, θ_2 is the smallest angle between the orthogonal complement of F with respect to u_1 and that of G with respect to v_1.
- and so on, until $k = dim(G) = q$.

3.2.2 How to compute the canonical correlation?

When two subspaces F and G are generated by columns of two matrices A and B, the canonical correlations can be computed by doing a singular value decomposition (SVD):

Let P_A and P_B be unitary bases for the subspaces spanned by A and B and let $M = P_A^T P_B$. Then, the canonical correlations between A and B are given by the singular values of M [16].

3.2.3 Martin Distance [25]

Here, I present the Martin distance, and I will use it to measure dissimilarity between two subspaces.

Definition 3.2 Given two subspaces F and G, the Martin distance between them with

respect to the principal angles $0 \leq \theta_1 \leq \theta_2 \leq ... \leq \theta_d \leq \pi/2$ is defined as:

$$d_M(F,G)^2 = -ln\left(\prod_{i=1}^{d} cos^2\theta_i\right) \tag{3.4}$$

In [25], you can find the proof that it is a metric.

3.2.4 Alternative Dissimilarities

In practice, if there are many Hankel matrices, computing the canonical correlations of their subspaces will be expensive. Therefore, we need some alternative dissimilarities so that it makes computation easier and faster. Two distances are more usual

$$d_1(H_1, H_2) = 2 - ||H_1 + H_2||_F^2 \tag{3.5}$$

where $H_i = \frac{H_i}{||H_i||_F}$ [6], and

$$d_2(H_1, H_2) = 2 - ||H_1 H_1' + H_2 H_2'||_F^2 \tag{3.6}$$

where $H_i = \frac{H_i}{||H_i H_i'||_F}$ [3].
where $||.||_F$ is the Frobenius norm. The first measure directly uses Hankel matrix H. In contrast to the first measure, the second measure uses the covariance matrix HH'. The matrix HH' is invariant to the direction in which the state changes [6].

Remark 3.3 d_1 and d_2 are used to estimate the dissimilarity between subspaces generated by the columns of two matrices. Strictly speaking, d_1 and d_2 are not metrics, however I will use the term distance or dissimilarity in the following.

3.3 Training in LVQ using Matrix based on subspace dissimiliarities

Now, I would like to describe two algorithms for learning the prototypes in LVQ. For both cases, the feature vectors will be orthogonal bases of subspaces associated with Hankel matrices.

For the first case, I will use the Martin distance as a dissimilarity measure. Since finding the derivative of d_M is too complex, I will use mGLVQ as an algorithm for learning prototypes.

For the second case, I will use the dissimilarities measures d_1 and d_2 with GLVQ, because d_1 and d_2 are differentiable.

3.3.1 Training Procedure using the Martin distance

In this case, I will use the Martin distance which is based on canonical correlations. Since finding the derivatives of canonical correlations with respect to prototypes is so complex, I will use the median variant of GLVQ which does not need any derivatives.

Training Procedure

Output: Prototypes $W_1, W_2, ..., W_M$

1. **Feature extraction:** Collect time series
2. **Hankel Matrices Assembly:** Construct a Hankel matrix for each time series
3. **PCA:** Find an orthogonal basis (with d element) for each Hankel matrix

$$H_i H_i' \approx P_i \Lambda_i P_i' \qquad i = 1, ..., N \tag{3.7}$$

where P_i and Λ_i are the eigenvalue and eigenvector matrices of the d largest eigenvalues associated with i-th time series, respectively.

4. **Dissimilarity Matrix (D):** Find Martin distance between each pair of orthogonal bases, it means

$$D(i,j) = d_M(P_i, P_j) \tag{3.8}$$

5. **mGLVQ:** Use the dissimilarity matrix (D) as the input for mGLVQ

Prediction

If we would like to predict the label of a new time series

$$z_1, z_2, ..., z_T$$

we should use the following procedure

Prediction Procedure

1. **Hankel Matrix:** Construct the Hankel matrix of the time series H.
2. **PCA:** Find an orthogonal basis (with d element) for the Hankel matrix H

$$HH' \approx P \Lambda P' \tag{3.9}$$

3. **Dissimilarity Vector (D):** Find Martin distance between P and all prototype.

$$W_s = min_i \{d_M(P, W_i)\} \tag{3.10}$$

where $W_1, W_2, ..., W_M$ are prototypes.

4. **Prediction:** The label of the new time series is equal to the label of s-th prototype.

3.3.2 Training Procedure for GLVQ using d_1(3.5), d_2(3.6)

From previous chapter, we know the updating rule of GLVQ

$$\triangle W^{\pm} = \eta . \frac{\partial sgd(\mu(P))}{\partial \mu(P)} . \frac{\pm 2 d^{\mp}(P)}{(d^{+}(P) + d^{-}(P))^2} . \frac{\partial d^{\pm}(P)}{\partial W^{\pm}} \tag{3.11}$$

for a given data point P.

To use this rule, I have to calculate derivatives of the dissimilarity measure with respect to prototypes. In this section I will use the dissimilarity measures d_1 and d_2. In the following parts, I will present the derivative of them with respect to prototypes.

Derivative of d_1 (3.5)

We suppose

$$d_1(W, P) = 2 - ||W + P||_F^2 \tag{3.12}$$

to find the derivative $\frac{\partial d(W,P)}{\partial W}$, consider

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1d} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2d} \\ w_{31} & w_{32} & w_{33} & \cdots & w_{3d} \\ \vdots & \vdots & \vdots & & \vdots \\ w_{r1} & w_{r2} & w_{r3} & \cdots & w_{rd} \end{bmatrix}$$

and $P = P(V)$ obtained from time series V via Hankelization

$$P(V) = \begin{bmatrix} v_{11} & v_{12} & v_{13} & \cdots & v_{1d} \\ v_{21} & v_{22} & v_{23} & \cdots & v_{2d} \\ v_{31} & v_{32} & v_{33} & \cdots & v_{3d} \\ \vdots & \vdots & \vdots & & \vdots \\ v_{r1} & v_{r2} & v_{r3} & \cdots & v_{rd} \end{bmatrix}$$

Thus, we can rewrite the dissimilarity measure d_1 as

$$d_1(W, P) = 2 - \sum_{i=1}^{r} \sum_{j=1}^{d} (w_{ij} + v_{ij})^2 \tag{3.13}$$

So, the derivative will be obtained as

$$\frac{\partial d(W, P)}{\partial W} = -2(W + P) \tag{3.14}$$

Derivative of d_2 (3.6)

For the second measure, we have

$$d_2(W, P) = 2 - ||WW' + PP'||_F^2 \tag{3.15}$$

We look for the following derivatives

$$\frac{\partial d(W,P)}{\partial W} = \begin{bmatrix} \frac{\partial d(W,P)}{\partial w_{11}} & \frac{\partial d(W,P)}{\partial w_{12}} & \cdots & \frac{\partial d(W,P)}{\partial w_{1d}} \\ \frac{\partial d(W,P)}{\partial w_{2,1}} & \frac{\partial d(W,P)}{\partial w_{22}} & \cdots & \frac{\partial d(W,P)}{\partial w_{2d}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial d(W,P)}{\partial w_{r1}} & \frac{\partial d(W,P)}{\partial w_{r2}} & \cdots & \frac{\partial d(W,P)}{\partial w_{rd}} \end{bmatrix}$$

For simplicity, we suppose $a_{ij} = [PP']_{ij}$. Now we can write

$$WW' + PP' =$$

$$\begin{bmatrix} \sum_{k=1}^{d} w_{1k}^2 + a_{11} & \sum_{k=1}^{d} w_{1k}w_{2k} + a_{12} & \cdot & \sum_{k=1}^{d} w_{1k}w_{rk} + a_{1n} \\ \sum_{k=1}^{d} w_{1k}w_{2k} + a_{21} & \sum_{k=1}^{d} w_{2k}^2 + a_{22} & \cdot & \sum_{k=1}^{d} w_{2k}w_{rk} + a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{k=1}^{d} w_{1k}w_{rk} + a_{r1} & \sum_{k=1}^{d} w_{2k}w_{rk} + a_{r2} & \cdot & \sum_{k=1}^{d} w_{rk}^2 + a_{rr} \end{bmatrix}$$

Let's define the following matrix

$$L^{ij} = \begin{bmatrix} 0 & \cdots & 0 & w_{1j} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & w_{2j} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & w_{(i-1)j} & 0 & \cdots & 0 \\ w_{i1} & \cdots & w_{i(j-1)} & 2w_{ij} & w_{i(j+1)} & \cdots & w_{id} \\ 0 & \cdots & 0 & w_{(i+1)j} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & w_{rj} & 0 & \cdots & 0 \end{bmatrix}$$

Thus, the derivative can be calculated according to

$$\frac{\partial d(W,P)}{\partial w_{ij}} = -2\sum_{kl} L_{kl}^{ij} \cdot [WW' + PP']_{kl}$$

Training Procedure

Now, we have the derivative of dissimilarity measures with respect to prototypes, and we can use the update rule. To learn the prototypes, we should follow the following algorithm

Training Procedure

1. **Feature extraction:** Collect time series
2. **Hankel Matrices Assembly:** Construct a Hankel matrix for each time series

3. **PCA:** Find an orthogonal basis (with d element) for each Hankel matrix

$$H_i H_i' \approx P_i \Lambda_i P_i' \qquad i = 1, ..., N \tag{3.16}$$

where P_i and Λ_i are the eigenvalue and eigenvector matrices of the d largest eigenvalues associated with i-th time series, respectively.

4. **GLVQ:** Update the correct winners W^+ and incorrect winner W^-

$$W^{\pm} = W^{\pm} \pm \eta . \frac{\partial sgd(\mu(P))}{\partial \mu(P)} . \frac{2d^{\mp}(P)}{(d^+(P) + d^-(P))^2} . \frac{\partial d^{\pm}(P)}{\partial W^{\pm}}$$

Prediction

If we would like to predict the label of a new time series

$$z_1, z_2, ..., z_T$$

we should use the following procedure

Prediction Procedure

1. **Hankel Matrices:** Construct the Hankel matrix of the time series H.
2. **PCA:** Find an orthogonal basis (with d element) for the Hankel matrix H

$$H H' \approx P \Lambda P' \tag{3.17}$$

3. **Dissimilarity Vector (D):** Find the distance between P and all prototype.

$$W_s = min_i \{d(P, W_i)\} \tag{3.18}$$

where $W_1, W_2, ..., W_M$ are prototypes.

4. **Prediction:** The label of the new time series is equal to the label of s-th prototype.

Remark 3.4 So far, we assumed that each class is generated by one LTI system. As we use LVQ for training, we can make this assumption softer, it means, each class can be generated by several LTI systems. The number of the systems for a class can be controlled by the number of prototypes.
However, we still have a restriction: Each time series is generated by one LTI system.

4 Clustering

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). In clustering, there is no label for objects, and the algorithm should be able to find groups automatically.

In this chapter, I will present three algorithms which I can use for the following chapter. At first, I will explain the Neural Gas algorithm. Then I will describe two median variants of LVQ: the median Neural Gas and K-mean.

4.1 Neural Gas (NG)

Suppose data points $x_i \in R^n, i = 1, ..., N$, are distributed according to an underlying distribution P. As Martinez described in [19], the goal of the NG algorithm is to distribute prototypes $w_i \in R^n, i = 1, ..., p$, such that these prototypes represent the distribution P_X as accurately as possible.

Hence, we need a measure for accuracy or error. The NG uses the following cost function

$$E_{NG}(w) = \frac{1}{2C(\lambda)} \sum_{i=1}^{p} \int h_\lambda(k_i(x, w)).d(x, w^i)P(dx) \tag{4.1}$$

where

$$d(x, y) = (x - y)^2 \tag{4.2}$$

is the squared Euclidean distance,

$$k_i(x, w) = |\{w_j | d(x, w_j) < d(x, w_i)\}| \tag{4.3}$$

is the rank of the prototypes sorted according to the distances. In other word, $k_i(x, w)$ is the number of prototypes which are closer than w_i to the data point x.

The function h_λ, known as neighborhood function, is

$$h_\lambda(t) = exp(-\frac{t}{\lambda}) \tag{4.4}$$

a Gaussian shaped curve with neighborhood range $\lambda > 0$. $C(\lambda) = \sum_{i=1}^{p} h_\lambda(k_i)$ is a normalization constant.

To optimize the cost function, we can use the gradient descent learning

$$\triangle w_i = \epsilon.h_\lambda(k_i(x_j, w)).(x_j - w_i) \tag{4.5}$$

where ϵ is learning rate. For its proof, we refer to [19].

Using this learning rule, the Neural Gas algorithm is as follows

On-line variant of Neural Gas

1. Initialize $\lambda(\gg 0)$
2. Initialize prototypes randomly
3. Randomly select a data point x
4. Determine the ranks of all prototypes $k_i(x,w)$
5. Update all prototypes according to the learning rule
6. Slowly decrease λ
7. Check the convergence condition, if it is not valid go to 3.

4.1.1 The Neural Gas for matrices

The Neural Gas for matrices will be the same as before, except that we do not have the Euclidean distance. However, we use the dissimilarity measure d_1. From the definition of d_1, it is based on the Frobinius norm, which is an extension of the Euclidean norm.

The only thing that I have to prove for validity of NG is that the gradient descent learning formula is still

$$\triangle W_i = \epsilon.h_\lambda(k_i(P_j,w)).(P_j - W_i) \tag{4.6}$$

where P_i and W_i are data points and prototypes, respectively.

Proof: The proof is similar to that one explained in [19]. For simplicity, I define the following notation

$$b_i = P + W_i$$

From the equation 3.14, we get

$$\frac{\partial d(P_j,W_i)}{\partial W_i} = -2(P_j + W_i).$$

Hence, if we assume that $C(\lambda) = 1$, we will get

$$-\frac{\partial E}{\partial W_i} = R_i + \int h_\lambda(k_i(P,W)).(P+W_i)P(dx)$$

where

$$R_i = -\frac{1}{2}\sum_{j=1}^{p}\int h'_\lambda(k_j(P,W)).d_j^2\frac{\partial k_j(P,W)}{\partial W_i}P(dx)$$

we should show that R_i vanished for each $i = 1,...,p$.

We can write the following relation for k_j

$$k_j(P,W) = \sum_{l=1}^{p} H(d_j^2 - d_l^2)$$

where $H(.)$ is the Heaviside function. The derivative of the Heaviside function is the delta distribution δ. Thus, we can rewrite R_i as

$$R_i = \int h'_\lambda(k_i(P,W)) d_i^2 b_i \sum_{l=1}^{N} \delta(d_i^2 - d_l^2) P(dx) - \sum_{j=1}^{p} \int h'_\lambda(k_j(P,W)).d_j^2 b_i \delta(d_j^2 - d_i^2) P(dx)$$

The only case that the integrands in the second term are not zero is for those data points P so that $d_j^2 = d_i^2$ (because $\delta(0) = 1$). For these data points we can write

$$k_j(P,W) = \sum_{l=1}^{N} H(d_j^2 - d_l^2) = \sum_{l=1}^{N} H(d_i^2 - d_l^2) = k_i(P,W)$$

and, hence, we obtain

$$R_i = \int h'_\lambda(k_i(P,W)) d_i^2 b_i \sum_{l=1}^{N} \delta(d_i^2 - d_l^2) P(dx) - \int h'_\lambda(k_i(P,W)).d_i^2 b_i \sum_{j=1}^{p} \delta(d_j^2 - d_i^2) P(dx)$$

Thus, NG is applicable for d_1. □

4.2 The Median Neural Gas (mNG)

There are some situations in which

- data are not embedded in a vector space and no continuous adapation is possible and / or
- the derivative of the distance function does not exist

In these cases, we can not use the original NG algorithm. Fortunately, the median variant of NG is applicable because it only needs the distance matrix of data points. The following expression in the whole section (4.2) are based on [14].

In the NG, since we have $w_i \in R^n$, optimizing the cost function is a continuous problem. But for the median NG, prototypes are chosen from the discrete set given by the training points, $w_i \in X = \{x_1, ..., x_N\}$. Hence, finding the prototypes will be a discrete optimization problem.

In the median NG, the cost function is

$$E = \sum_{i=1}^{p} \sum_{j=1}^{N} h_\lambda(k_{ij}).d(x_j, w_i) \tag{4.7}$$

where k_{ij} is the rank of the prototype w_i for the data point x_j. For given data point x_j, the values of k_{ij} constitutes a permutation of $\{0, 1, ..., p-1\}$, and they are treated as latent variables.

In the following parts, at first I will describe the median variant of Neual Gas, and

then I will present the proof of convergence.

4.2.1 Algorithm

To find the prototypes, the cost function E should be optimized within the set X^p, given by the training data.

The cost function can be interpreted as a function depending on k_{ij} and w. Hence, an iterative algorithm can be hired with two adaption stages. At first, E is optimized with respect to k_{ij} and then with respect to the prototypes w.

The Median Neural Gas

Input: data points, NumberOfIteration
Output: Prototypes

1. t=0
2. Initialize the prototypes by data points randomly
3. Determine

$$k_{ij} = k_i(x_j, w) = |\{w_l | d(x_j, w_l) < d(x_j, w_i)\}|$$

   as the rank of prototype w_i.
4. Based on the latent variables k_{ij}, set

$$w_i = x_l$$

   where

$$l = argmin_{l'} \sum_{j=1}^{N} h_\lambda(k_{ij}).d(x_j, x_{l'})$$

5. t=t+1
6. if $t < NumberOfIteration$ then go to 3. Otherwise, End.

Similar to the EM algorithm, at first the latent variables k_{ij} are updated (the prototypes are fixed), then the prototypes are updated (the latent variable are fixed).

Note:

For roughly ordered maps, we can restrict the potential candidate x_l to data points mapped to a neighborhood of i, because it will speed up training.

4.2.2 Convergence [14]

The median NG optimizes $E = E(w)$ by consecutive optimization of the latent variables $k_{ij}(w)$ and prototypes w. The values k_{ij} will be unique for given w, and the optimum

w will also be unique for given k_{ij} by introducing an order.

Consider the function

$$Q(w', w) = \sum_{i=1}^{p} \sum_{j=1}^{N} h_\lambda(k_{ij}(w)).d(x_j, w'_i) \tag{4.8}$$

where w are given values of the prototypes and w' are the new values of the prototypes based on $k_{ij}(w)$.

Note that $E(w) = Q(w, w)$ and for convergence we should prove that

$$E(w) = Q(w, w) \leq Q(w', w') = E(w')$$

Since w' are the optimum assignment of the prototypes given the latent variables k_{ij}, it can be concluded that $E(w) = Q(w, w) \leq Q(w', w)$. On the other hand, we can write $Q(w', w) \leq Q(w', w') = E(w')$ because $k_{ij}(w')$ are the optimum assignment of the latent variable given the new values of prototypes w'. Hence, we get

$$E(w) = Q(w, w) \leq Q(w', w) \leq Q(w', w') = E(w') \tag{4.9}$$

So, the cost function will decrease in each iteration.

Since there exists a finite number of different values k_{ij} and the assignments are unique, the median NG converge in a finite number of steps toward a fixed point w^*.

4.3 Median k-means

Some papers use a modified version of k-means. This method selects prototypes from data points, similar to the median NG. Here, I will describe it, according to [3].

Assume we have data points $x_j \in R^n, j = 1, ..., N$. The following algorithm will find the prototypes $w_i, i = 1, ..., p$.

k-means

Input: Training Set, NumberOfIteration
Output: Prototypes

1. $t := 0$
2. Initialize prototypes randomly by data points
3. Determine

$$s = min_i d(x_j, w_i) \quad where \quad i = 1, ..., p \tag{4.10}$$

the winner for each data point x_j, and assign it to the cluster w_s

4. Select a data point x_i from the cluster w randomly, and Let

$$D_i = \{d_{i1}, d_{i2}, ..., d_{in_w}\}$$

be the dissimilarity scores for all data point in cluster w with respect to x_i. Then, the data point that has the dissimilarity score closest to μ_w (the mean of the elements of D_i) is selected as the center of this cluster.

5. $t := t + 1$.

6. if $t < NumberOfIteration$, then go to 3. Otherwise, End.

Remark 4.1 For our case, we can directly use Hankel matrices with the median Neural Gas (or K-means). However, for the Neural Gas we just use Hankel matrices to generate d-dimension subspaces. Thus we need to find a way to estimate d. (like cross validation)

5 Set of Hanklets

So far, I have assumed that a time series is an output of a LTI system. It means we used one LTI system to approximate a time series, which is inappropriate for many cases.

The first solution for this problem is to approximate a time series with a set of LTI system, instead of one LTI system. Here, I will use the bag-of-words model to describe a time series with a bag-of-Hanklets (BoHk). The method explained in this chapter is based on [3].

The bag-of-words model is a simplifying representation used in natural language processing [20]. In this model, a text is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.

The bag-of-words model is commonly used in methods of document classification, and it is mainly used as a tool of feature generation. The frequency of each word is used as a feature for training a classifier. Here,I will also use the bag-of-words model to generate features. The only requirement is to define a codebook (or dictionary).

In our case, as I want to use LTI systems, according to [3], Hankle matrices of LTI systems will be equivalent to words. The codebook will be constructed by clustering dynamical systems.

In the following sections, at first I will describe how to build words (LTI systems) in pre-processing step, then I will explain how to build a codebook by clustering algorithms. In the third section, I will describe the Bayes' classifier. Because the outputs of a LTI system will be different for different initial values, we need an assignment algorithm to determine which LTI system (word) is responsible for a sequence. At last, I will describe an algorithm for classification.

5.1 Pre-processing

Now, we suppose that each time series is represented by a sequence of overlapping temporal window.

The following procedure will be done for each time series in training set

Building Words

1. $t = 1$.
2. Divide t-th time series into m smaller sets with equal length

$$\{y_1^t, y_2^t, ..., y_N^t\} \rightarrow s_1 = \{y_1^t, y_2^t, ..., y_T^t\}, ...$$

$$..., s_m = \{y_{N-T+1}^t, y_{N-T+2}^t, ..., y_N^t\}$$

3. Construct a Hankel matrix H_k^t for each set s_k

$$H_1^t, H_2^t, ..., H_m^t$$

where H_i plays the role of words

4. $t := t + 1$.
5. If $t \leq NumberOfTimeSeris$, then go to 2. Otherwise, End.

The procedure yields a set of Hankel matrices, known as Hanklets. These Hanklets play the role of words.

5.2 Clustering Hanklets

After the preprocessing, we will get many Hanklets. However, the number of LTI systems that generate the Hanklets are limited. To find the LTI systems, we generate a codebook which contains a limited number of Hankel matrices associated to LTI systems.

Clustering algorithms will be used to generate the codebook. The codebook contains center of clusters.

Remark 5.1 As I am using Hanklets directly (I did not use Hankel matrices to generate subspaces), I should use the median Neural Gas or K-means (not Neural Gas) for clustering .
If we would like to use the Neural Gas, we have to use PCA for each Hanklet to generate a d-dimensional subspace.

For clustering, one of the dissimilarity measures d_1 from equation 3.5 or d_2 from equation 3.6 can be selected. The output of clustering algorithms will be some prototypes which are centers of clusters. Hence, the set of prototypes will be used as a codebook.

5.3 Labeling Hanklets

After generating a codebook, we should determine for each of the Hanklets the representing cluster. That is equivalent to determine which words correspond to Hanklets. Here, we will use the Bayes' classifier to assign each Hanklet to a cluster (word).

5.3.1 Bayes' Classifier

The Bayes' classifier is a probabilistic classifier based on applying Bayes' theorem [15].

$$p(A|B) = \frac{p(A)p(B|A)}{p(B)} \tag{5.1}$$

where A and B are two events, and where $p(A)$ and $p(B)$ are the probabilities of observing A and B. $p(B|A)$ is the probability of observing event B given that A is true.

Assume x is an observation and $L_K = \{1, ..., K\}$ are set of labels playing the role of B and A, respectively. The Bayes' classifier assigns x to the class that is most probable. In other words, the label of x will be predicted as k if

$$k = argmax_{i \in L_K} \left(p(i|x)\right) \tag{5.2}$$

From Bayes' theorem

$$k = argmax_{i \in L_K} \left(\frac{p(i)p(x|i)}{p(x)}\right) \tag{5.3}$$

where $p(i)$ and $p(x|i)$ are the prior distribution of class i and conditional density function of x given class i.

Since $p(x)$ doesnot depend on the label, the denominator is effectively constant, and we get

$$k = argmax_{i \in L_K} \left(p(i).p(x|i)\right) \tag{5.4}$$

5.3.2 Bayes' Classifier for Hanklets

From the above formula, the Bayes' classifier needs prior distribution of classes $p(i), i = 1, ..., K$ and likelihood function $p(x|i)$.

In our context for a given Hanklet H, x represents the dissimilarity between H and the center of each cluster.

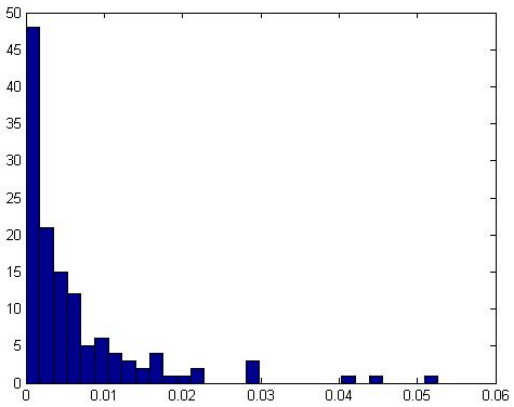

Figure 5.1: The histogram of dissimilarity scores for a typical cluster in the dictionary of Hankelets resembles a Gamma distribution

Figure 5.1 shows a histogram of the dissimilarity scores with respect to the center of a cluster. As you can see in this figure, the distribution is close to a Gamma distribution

so that a large number of Hanklets has very small dissimilarities and the number of Hanklets exponentially decrease for increasing dissimilarities. Thus, we approximate the posterior distribution of cluster w by a Gamma distribution

$$p(d|w) = \begin{cases} \frac{a^b d^{b-1}}{(b-1)!} e^{-ad} & ford \geq 0 \\ 0 & otherwise \end{cases}$$

the mean $\mu_w = \frac{b}{a}$ and variance $\sigma_w^2 = \frac{b}{a^2}$ estimated from the data. $p(d|w)$ plays the role of $p(x|i)$ in equation 5.4. Furthermore, each cluster w has a prior distribution $P(w)$ where

$$P(w) = \frac{Numberof Hanklets in\, Cluster w}{Total Number of Training Hanklets} \tag{5.5}$$

Thus, from equation 5.4 for the Bayes' classifier, we get

$$k = argmax_{w \in L_K} (P(w).P(d|w)) \tag{5.6}$$

where L_K is set of labels of clusters.

5.4 Bag-of-Hanklets

Now, we have a codebook and we know the clusters that Hanklets belong to. Each time series is represented with a Bag-of-Hanklets (HoHk), it means a histogram of words from the codebook of Hanklets.

Building Words

For a given time series $\{y_i\}_{i=1}^N$, we have a set of Hanklets

$$H_1, H_2, ..., H_m$$

The BoHk of it will be

$$[f_1, f_2, ..., f_K]^T$$

where f_i is frequency of cluster i in this time series (the number of Hanklets which belong to cluster i) and where K is number of clusters.

Hence, we get a feature vector (BoHk) for each time series, and we can use them to train a classifier.

5.5 Classification Algorithm

In the following, I summarize all steps

Classification Algorithm

Training

1. **Feature Extraction:** Collect time series
2. **Pre-Processing:** Construct a set of Hanklets for each time series
3. **Clustering:** Make a codebook (or dictionary) for hanklets
4. **Bayes' Classifier:** assign each hanklets to a cluster
5. **BoHk:** Make a histogram for each time series
6. **GLVQ:** Use BoHks as feature vector

Remark 5.2 Selecting the appropriate set of features (in feature generation) is very important. It means that the features should encode as much class-discriminatory information, by measuring their value for a given pattern, to be able to predict the label of the pattern[17]. Since the BoW representation ignores temporal order, the importance of features generation step increases. In some cases, to get a good set of features, this step will be time consuming.

6 Experiments

In this chapter, I used the methods which I explained in my thesis, and I report the result.

6.1 Algorithms in Chapter 3

I applied the classification algorithm described in chapter 3, and I report the results for different number of prototypes.
To use the algorithms in the chapter 3, we should determine two parameters

- L: The number of measurement vectors in a column of Hankel matrices
- d: Dimension of state space

For all experiments, L and d are obtained by n-fold cross validation, for each method in use separately.

6.1.1 PenDigit Dataset (UCI)

It is a digit database by collecting 250 samples from 44 writers. Here, I select 300 samples for training and the rest of them are for testing.

To make this data set, they used a WACOM PL-100V pressure sensitive tablet with an integrated LCD display and a cordless stylus. The tablet sends x and y tablet coordinates of the pen at fixed time intervals (sampling rate) of 100 miliseconds.

Each data point contains

- a time series of length 8: the coordinates of the pen in 8 steps. it means

$$(s_1, s_2, ..., s_8)$$

  where $s_i = [x_i, y_i]^T \in \mathbb{R}^2, i = 1, ..., 8$
- a label $l \in \{0, 1, ..., 9\}$

You can see one sample in figure 6.1

mGLVQ with CC

I applied the mGLVQ algorithm with Martin distance (based on canonical correlations) and I set the parameters to

- $L = 5$.
- $d = 3$.

At first, I supposed one prototype per class, and I got the prototypes in figure 6.2

The accuracy of the classification was 77 percent. The confusion matrix is reported in table 6.1

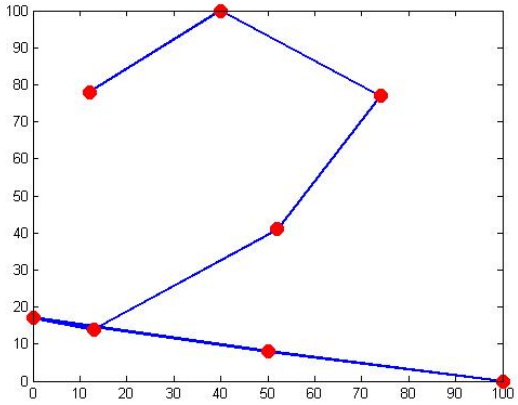

Figure 6.1: A data point with label 2

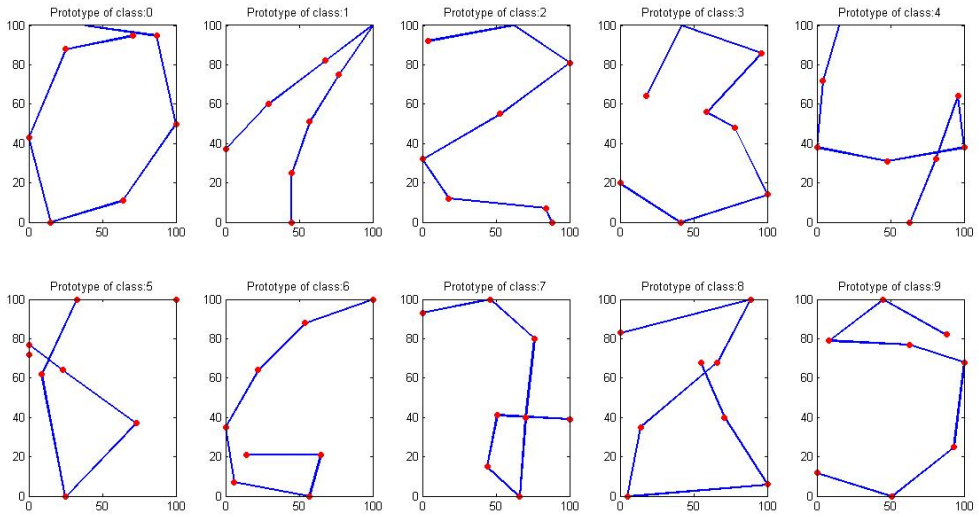

Figure 6.2: Prototypes of Pendigit dataset: mGLVQ with 1 prototype per class

GLVQ

As I said in chapter 3, I used GLVQ with the dissimilarity measure d_1. In this case, the parameters are equal to

- $L = 7$.
- $d = 1$.

For one prototype per class, the accuracy of classification is 81 percent. The table 6.2 is represented the confusion matrix.

Conclusions

As you can see from tables 6.1 and 6.2

	0	1	2	3	4	5	6	7	8	9
0	100	0	0	0	0	0	0	0	0	0
1	4	58	4	3	1	0	1	26	3	0
2	0	2	94	0	0	0	0	2	2	0
3	0	1	1	95	0	0	0	0	0	3
4	1	1	2	0	91	0	2	0	0	3
5	0	0	11	9	0	56	0	3	0	22
6	4	0	0	0	4	0	92	0	0	0
7	1	14	1	3	0	0	0	76	2	2
8	4	1	2	4	0	46	2	1	39	0
9	1	3	10	9	1	9	2	0	0	64

Table 6.1: Confusion matrix of Pendigit Dataset: mGLVQ with 1 prototype per class

	0	1	2	3	4	5	6	7	8	9
0	90	1	0	0	1	0	5	0	3	0
1	0	60	27	5	0	0	1	1	0	6
2	0	3	96	0	0	0	0	0	0	0
3	0	2	0	98	0	0	0	0	0	0
4	0	1	0	0	91	0	6	0	0	2
5	0	1	0	11	0	59	4	0	0	25
6	0	0	0	0	2	0	97	0	0	0
7	0	13	0	4	1	0	0	82	0	0
8	17	2	4	2	0	5	1	5	63	1
9	0	13	1	3	8	0	0	0	0	74

Table 6.2: Confusion matrix of Pendigit Dataset: GLVQ with 1 prototype per class

- In mGLVQ, the variations of accuracies in different classes is more. For example, the minimum and maximum values of accuracy (38 and 100 percent) happen in mGLVQ.
- Classifying the numbers 1,5,8 and 9 are more difficult for the classifiers.

Thus, to increase the accuracy, I increased the numbers of prototypes such that the numbers of prototypes are as the table 6.3

# of protos	0	1	2	3	4	5	6	7	8	9
30 protos	2	4	2	2	2	4	2	2	4	6
50 protos	4	6	4	4	4	6	4	4	6	8

Table 6.3: Number of prototypes per class

The table 6.4 shows the accuracy of classifiers if the total number of prototypes are 30 and 50, respectively.

From the table 6.4, we can conclude

Algorithms	# of Protos 10	30	50
mGLVQ (CC)	77	88	90
GLVQ (AI)	81	87	88

Table 6.4: Accuracy of algorithms for PenDigit dataset

- If the number of prototypes are low (1 per class), the GLVQ gives us better result.
- If the number of prototypes are high, the mGLVQ gives better result.

6.1.2 Libras Dataset (UCI)

The dataset (movement libras) contains 15 classes of 24 instances each, where each class references to a hand movement type in LIBRAS. We divide them into two subsets with the same number of elements, one for training and another for testing.

In the video pre-processing, a time normalization is carried out selecting 45 frames from each video, in according to an uniform distribution. In each frame, the centroid pixels of the segmented objects (the hand) are found, which compose the discrete version of the curve F with 45 points. All curves are normalized in the unitary space.

Each data point contains

- a time series of length 45

$$(s_1, s_2, ..., s_45)$$

  where $s_i = [x_i, y_i]^T \in \mathbb{R}^2, i = 1, ..., 45$
- a label $l \in \{1, 2, ..., 15\}$

mGLVQ (CC)

To use the first algorithm of chapter 3, I set the parameters to

- $L = 33$.
- $d = 3$.

For one prototype per class, the accuracy of classification is 57 percent. The table 6.5 shows the confusion matrix.

GLVQ

I set the parameters equal to

- $L = 41$.
- $d = 2$.

By using one prototype per class, the accuracy is 59 percent, and I report the confusion matrix in table 6.6

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	67	33	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	67	0	25	0	0	0	0	0	0	0	8	0
5	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0
6	0	0	0	17	0	75	0	0	0	0	0	0	0	8	0
7	0	0	0	42	17	0	33	0	0	0	0	0	0	8	0
8	0	0	0	0	0.08	0	0	67	0	8	0	0	0	8	8
9	0	17	17	0	0	8	17	0	8	0	17	0	0	8	8
10	0	0	33	0	0	0	8	0	0	58	0	0	0	0	0
11	0	42	0	0	0	0	0	0	0	0	25	0	33	0	0
12	0	0	8	0	0	0	0	0	0	17	0	75	0	0	0
13	0	75	0	0	0	0	0	0	0	0	17	0	8	0	0
14	0	0	0	17	8	0	0	0	0	0	0	0	0	75	0
15	0	0	0	8	67	0	0	8	0	0	0	0	0	17	0

Table 6.5: Confusion Matrix of Libras Dataset: mGLVQ with 1 proto

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	75	25	0	0	0	0	0	0	0	0	0	0	0	0	0
2	8	33	0	0	0	25	0	0	0	8	8	0	0	0	17
3	0	0	92	0	0	8	0	0	0	0	0	0	0	0	0
4	0	0	17	58	0	0	8	0	0	0	0	8	0	8	0
5	0	0	0	0	92	0	0	0	0	0	0	0	0	0	8
6	0	17	8	17	0	42	0	0	0	8	0	0	0	8	0
7	0	0	0	8	8	0	75	0	0	0	0	8	0	0	0
8	0	0	0	0	0	0	8	75	0	8	0	0	8	0	0
9	17	8	0	0	0	0	0	0	75	0	0	0	0	0	0
10	8	0	8	0	0	0	0	0	0	42	0	42	0	0	0
11	0	0	0	0	0	0	0	8	0	0	17	8	50	0	17
12	0	0	8	8	8	0	8	0	0	0	0	67	0	0	0
13	8	8	0	0	0	0	0	0	0	8	8	8	50	0	8
14	0	0	0	0	0	0	17	8	0	0	0	0	0	75	0
15	0	0	0	0	33	0	25	17	0	8	0	0	0	0	17

Table 6.6: Confusion Matrix (percentage) of Libras Dataset: GLVQ with 1 proto

Conclusion

From table 6.5 and 6.6,

- The variations of accuracy for different classes in mGLVQ are higher.
- the mGLVQ can not classify the class 15 correctly at all.

To increase accuracy, I increase the numbers of prototypes , and I set them to 3 and 5 per class, respectively. The result can be found in table 6.7

	# of Protos		
Algorithms	15	45	75
mGLVQ (CC)	57	64	72
GLVQ (AI)	59	64	68

Table 6.7: Accuracy of algorithms for Libras dataset

6.1.3 Comparison

In this section, I would like to use the result that I get from two data sets to compare two algorithms.

- If the number of prorotypes is much smaller than the number of elements in training set, the senond method (GLVQ with d_1) will give better result. However, If the number of prototypes are comparable to the number of elements in training set, the first method (mGLVQ with d_M) will give better result. Why?
  - If the number of prototypes is low, the role of learning strategy will be important. Since the mGLVQ is restricted (prototypes should be data points), the GLVQ will give better result.
  - If the number of prototypes is high, the role of dissimilarity measure will be more important. d_1 just estimate the dissimilarities between subspaces, but d_M is a metric to measure the dissimilarities between subspaces.
- The variations of accuracy for different classes in mGLVQ are higher.

Remark 6.1 To increase the accuracy of classification for GLVQ with the dissimilarity measure d_1, we can use the kernel trick introduced in [27] and [28]. The only thing we need to do is to convert matrices into vectors

$$P = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1d} \\ v_{21} & v_{22} & \cdots & v_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ v_{m1} & v_{m2} & \cdots & v_{md} \end{bmatrix} \rightarrow P' = \begin{bmatrix} v_{11} \\ v_{21} \\ \vdots \\ v_{m1} \\ v_{12} \\ v_{22} \\ \vdots \\ v_{m2} \\ \vdots \\ v_{1d} \\ \vdots \\ v_{md} \end{bmatrix}$$

7 Discussion

As you have seen, at first I supposed that time series in a certain class come from an LTI system. Later, I made this assumption softer, it means each class could come from several LTI systems. However, we still had a restriction that each time series has to be generated by one LTI system. In chapter 5, we released this restriction, and we supposed that each time series could come from a set of LTI systems.

Here, I present two topics that need to work in future:

- To make a feature vector from the set of Hanklets, we used Bag-of-Word model. But this model does not take into account the temporal order of words. To get a good accuracy, we have to use an appropriate set of features in feature extraction step. However, it is sometimes a time consuming process and it limits the speed of feature extraction. One way to prevent this problem is to use some models, instead of BoW model, which take into account the temporal order of words. For instance, Presti [6] suggested to use Hidden Markov Model (HMM). She trains a HMM for each class.
- In my thesis, I used Hankel matrices to construct new feature vectors. Its simple structure is able us to capture the temporal information of time series which come from the LTI systems. By using more complex structure, maybe we could work with the time series which come from more complex systems.

Bibliography

[1] T. Kim, J. Kittler, and R. Cipolla. *Discriminative learning and recognition of image set classes using canonical correlations*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 29 (6), pp.1005-1018, 2007.

[2] B. Li, M. Ayazoglu, T. Mao, O. Camps, and M. Sznaier. *Activity recognition using dynamic subspace angles*. In: 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.3193-3200, IEEE (2011).

[3] B. Li, O. Camps, and M. Sznaier. *Cross-view Activity recognition using Hankelets*. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.1362-1369, IEEE (2012).

[4] M. Ayazoglu, B. Li, C. Dicle, M. Sznaier, and O. Camps. *Dynamic Subspace-Based Coordinated Multicamera Tracking*. In: 2011 IEEE International Conference on Computer Vision (ICCV), pp. 2462-2467, IEEE (2011).

[5] D. Lai, and G. Chen. *Dynamical Systems Identification from Time-Series Data: A Hankel Matrix Approach*. Mathl. Comput. Modeling, Vol 24, No. 3, pp. 1-10, 1996.

[6] L. Presti, M. Cascia, S. Sclaroff, and O. Camps. *Gesture Modeling by Hanklet-based Hidden Markov Model*. 12th Asian Conference on Computer Vision (ACCV), pp. 529-546, 2014.

[7] A. Ravichandran, R. Chaudhry, and R. Vidal. *View-Invariant Dynamic Texture Recognition using a Bag of Dynamical Systems*. Proc. IEEE Conference Computer Vision and Pattern Recognition (CVPR), pp.1-6, 2009.

[8] R. Mahmoudvand, and M. Zokaei. *On the singular values of the Hankel matrix with application in singular spectrum analysis*. Chilean Journal of Statistics, Vol. 3, No. 1, pp. 43-56, 2012.

[9] T. Villmann. *Sobolev Metrics for Learning of Functional Data - Mathematical and Theoretical Aspects*. pp. 1-13, 2006.

[10] J. Lee, and M. Verleysen. *Generalization of the L_p norm for time series and its application to self-organizing maps*. 5th Workshop on Self-Organizing Maps (WSOM), pp.733-740, 2005.

[11] D. Nebel, B. Hammer, K. Frohberg, and T. Villmann. *Median variants of learning vector quantization for learning of dissimilarity data*. Neurocomputing, Vol. 169, pp.295-305, 2015.

[12] D. Nebel, B. Hammer, and T. Villmann. *A Median variants of Generalized Learning Vector Quantization*. Neural Information Processing, Vol. 8227, pp. 19-26, 2013.

[13] L. Presti, and M. Cascia. *Using Hankel Matrices for Dynamics-based Facial Emotion Recognition and Pain Detection*. In CVPR, pp.26-33, 2015.

[14] M. Cottrell, B. Hammer, A. Hasenfuss, and T. Villmann. *Batch and median neural gas*. Neural Networks, Vol. 19, Issues 6-7, pp.762-771, 2006.

[15] C. Bishop. *Machine Learning and Pattern Recognition*. Springer, 2006.

[16] A. Bjorck, and G. H. Golub. *Numerical methods for computing angles between linear subspaces*. Mathematics of Computation, Vol. 27, No. 123, pp. 579-594,

1973.

[17] S. Theodoridis. *Machine Learning - A Bayesian and Optimization Perspective*. Elsevier, 2015.

[18] M. T. Hagan, H. B. Demuth, M. Beale, O. Jesus. *Neural Network Design*. 2nd Edition, 2014.

[19] T. Martinez, S. G. Berkovich, and K.J. Schulten. *Neural-gas Network for vector quantization and its application to time-series prediction*. IEEE Transactions on Neural Networks, Vol.4, No. 4, pp. 558-569, 1993.

[20] H. Zellig. *Distributional Structure*. IEEE Transactions on Neural Networks, Word, Vol. 10, Issue 2-3, pp. 146-62, 1954.

[21] M. Kaden, M. Lange, D. Nebel, M. Riedel, T. Geweniger, and T. Villman. *Aspects in Classification Learning - Review of Recent Developments in Learning Vector Quantization*. Foundations of Computing and Decision Sciences, Vol. 39, No. 2, pp. 79-105, 2014.

[22] A. Sato and K. Yamada. *Generalized learning vector quantization*. Advances in Neural Information Processing Systems 8, pp. 423-429, 1996.

[23] A.P. Dempster, N.M. Laird, D.B. Rubin. *Maximum likelihood from incomplete data via the EM algorithm*. J.R. Stat. Soc. Ser. B 39 (1977) 1-38.

[24] L. Bottou, and O. Bousquet. *The Tradeoffs of Large Scale Learning*. Advances in Neural Information Processing Systems, 20:161-168.

[25] Martin R.J. *A metric for ARMA processes*. IEEE Transactions on Signal Processing, Vol.48, No.4, pp.1164-1170, 2000.

[26] R. Stuart, and P. Norvig. *Artificial Intelligence: A Modern Approach (2nd ed.)*. Prentice Hall, 2003.

[27] Qin. A.K., Suganthan, P.N. *A novel kernel prototype-based learning algorithm*. In: Proc. of the 17th International Conference on Pattern Recognition (4). pp. 621-624, 2004.

[28] F.M. Schleif, T. Villmann, B. Hammer, P. Schneider, and M. Biehl. *Generalized derivative based kernelized Learning Vector Quantization*. Intelligent Data Engineering and Automated Learning âĂŞ IDEAL 2010, Vol. 6283, pp. 21-28, 2010.

[29] Robbins, H.; Montro, s. *A Stochastic Approximation Method*. The Annals of Mathematical Statistics. 22 (3), 1951.

[30] R. Sanchez Pena, and M. Sznaier. *Robust Systems Theory and Applications*. John Wiley Sons, New York, NY, 1998.

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.
Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, im September 2016